UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/727,327	11/29/2000	Marian Montequinfela MacCormack	50886-03USPX	2273

7590 06/02/2004

Andre M. Szuwalski
Jenkens & Gilchrist, P.C.
Suite 3200
1445 Ross Avenue
Dallas, TX 75202-2799

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2124	9

DATE MAILED: 06/02/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/727,327	MACCORMACK, MARIAN MONTEQUINFELA	
	Examiner	Art Unit	
	Tuan A Vu	2124	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 March 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-21* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-21* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the application filed March 29, 2004.

Claims 20-21 are added and claims 1-21 have been submitted for examination.

Specification

2. The abstract of the disclosure is objected to because the United Kingdom Patent Application Number provided (pg. 7, bottom 3rd para) does not match the actual copy of Foreign Patent Application submitted in the IDS papers filed as of 5/14/2001, whose application number is 9928340.0. Correction is required. See MPEP § 608.01(b).

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 1-4, 6-17, 19, and 20-21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272 (hereinafter Cahill), in view of Brooks et al., USPN: 5,819,097 (hereinafter Brooks).

As per claim 1, Cahill discloses a method of generating code listing from an object code sequence comprising a section data including a plurality of program instructions, said section associated a relocation section including at least one relocation instruction (e.g. *tag table 218, text relocs 305 , data relocs 306* - Fig. 3; col. 7, lines 26-35) used at link time to modify the object code to generate an executable program (Fig. 1 – Note: linker and object code implicitly discloses modify object code at link time to generate executable), the method comprising:

for each location in the section data determining if that location has a relocation instruction associated therewith (e.g. *relocation tags* - col. 3, lines 18-25; col. 6, line 56 to col. 7, line 35 – Note: based of derived PIOF from scanning a object module section scanning, locating tags for relocation type is equivalent to determining if relocation instruction is associated with a section data);

deriving relocation additional information concerning the section data (e.g. *tag type* - col. 7, lines 25-35; *TYPE, NEXT-INST, TAGS* – Fig. 4); and

generating the reversed object code listing for that object code modification instruction and additional information, such listing comprising the modified object code that was derived from said additional information (e.g. Fig. 2, 5, 7A-D).

But Cahill does not explicitly specify reading said relocation instruction identified for each location in the section data. However, Cahill discloses disassembling process to recompose the original object code based on scanning sections of the object module and readjust each instruction-related location data, and *jump table* relocation instruction, such modification derived from additional information like tag, target address, target label (e.g. Fig. 4-5; col. 9, line 45 to col. 10, line 53). Hence Cahill has implicitly disclosed reading relocation instruction identified at a location in the section data.

Nor does Cahill specify generating the source code listing in lieu of reversed object code for the location in the section data, such listing being derived from said relocation instruction and additional relocation information. However, Cahill associates disassembling of object sections with special structures including pointers, labels or tags/references to instructions or data associated with a symbol table (Fig. 3-4; col. 6, lines 55-65). In a method analogous to that of

Cahill using decompilation or unlinking of a compiled object program (*object program 24*, *unlinked object 54* -Fig. 2), Brooks discloses unlinking the object program back into source code (e.g. Fig. 5) using additional information such as instruction table and symbol table, memory address readjustment of values, and tags values replacements (e.g. col. 7, line 37 to col. 8, line 8; col. 8, line 66 to col. 9, line 21; Fig. 6 – Note: at binding time, the fact of finding a temporary value in the object code data and replacing it with a concrete value is similar to relocation operation) similar to the relocation additional information as taught by Cahill. It would have been obvious for one of ordinary skill in the art at the time the invention was made to implement the disassembling process as taught by Cahill so that the recreation of source code is based on relocation identification and additional relocation information as disclosed by Brooks, because such source code being recreated can serve as human-remodifiable, for user accommodating graphical display and editing, and reusable program and obviating compiler resources because the source code thus re-generated would not require making change to the compiler or the hardware controller (Brooks: col. 1, lines 39-59; col. 2, lines 50-59); and further because decompilation to source code using the available results from disassembling object codes as suggested by Cahill and furthered by Brooks would take full advantage of resources in a opportunistic manner, thus enabling reuse of high-level code in other platforms applications as intended by Cahill.

As per claim 2, this source code limitation would have been obvious in view of the rationale used in claim 1 above using Brooks' teaching.

As per claims 3-4, Cahill discloses object code sequence being part of the object code module (e.g. col. 3, lines 18-25) and part of an executable program (Note: text and data sections in a object module implicitly teaches object code sequence being part of the executable program)

As per claim 6, Cahill discloses relocation instructions for performing calculations using a stack (e.g. Fig. 4 – Note: For a skill in the art, the fact of using linked list and retrieving data from the linking process implicitly discloses a stack usage to pop/push data back and forth for processing the linked list; *tags stack* – col. 9, lines 65-67; Fig. 5)

As per claim 7, Cahill discloses arithmetic expression from branch instruction generated from source code (e.g. Fig. 5; *jump tables* - col. 9, line 45 to col. 10, line 19 – Note: since object code is inherently derived from original source code, the branch instructions associated with jump tables being defined from source code are implicitly disclosed and the arithmetic calculations as to replace relative address with absolute location value at link time is inherent to all relocation operation).

As per claim 8, Cahill discloses assembler directives (e.g. *PIAI, checkpoint instructions* – col. 11-12; col. 12, table 6-1, 6-2; assembly code instruction *400* – Fig. 4).

As per claim 9, Cahill discloses operands values defined in the source code (e.g. col. 9, lines 12-36 Note: since object code is inherently derived from original source code, operands instructions *tuples* being defined from source code are implicitly disclosed).

As per claim 10, Cahill disclose event information (e.g. *signal handler* – col. 15, line 35 to col. 16, line 4).

As per claim 11, Cahill discloses a lister for generating reversed object code listing from an object code sequence comprising program instructions associating a relocation instruction therewith; the lister comprising:

a instruction reader (e.g. col. 6, lines 55-65);

a relocation reader (*relocation tags* - col. 3, lines 18-25; col. 6, line 56 to col. 7, line 35);

a relocation identifier (*relocation tags* - col. 3, lines 18-25; Fig. 5);

a disassembler for disassembling said program instructions to generate reversed object code and disassembling additional information (e.g. *tags* - col. 7, lines 25-35) from said relocation instruction (Fig. 2, 5; 7A-D).

But Cahill does not specify generating the source code display in lieu of reversed object code from disassembling program instructions, such source code listing being derived from said relocation instruction and additional relocation information and being in human-readable form. But this limitation has been addressed in claim 1 above.

As per claim 12, Cahill discloses offset determining means for determining offset from relocation instruction (e.g. col. 6, line 56 to col. 7, line 35 – Note: offset information used in standard relocation operation at linking time is implicitly disclosed because no relocation can be effected without a form of offset address to be replaced; col. 11, lines 24-44).

As per claim 13, Cahill discloses a type field (e.g. *Type 430* - Fig. 4).

As per claim 14, Cahill discloses an operand, an operator (e.g. *tuple* – col. 9, lines 9-21; *jump table* – col. 9, line 40 to col. 10, line 54 – Note: branch instructions inherently includes operands and operator for address adjustment),

an event (col. 15, line 35 to col. 16, line 4); and

a directive (e.g. *PIAI, checkpoint instructions* – col. 11-12; col. 12, table 6-1, 6-2; assembly code instruction *400* – Fig. 4).

As per claim 15, see claim 7 rejection.

As per claim 16, see rejection of claim 6.

As per claim 17, Cahill discloses selecting of event handler in resolving reallocation of addresses (*signal handler* – col. 15, line 35 to col. 16, line 4) but does not expressly disclose event calculator. However, Cahill discloses selectively calling of handler according to set bit (e.g. Fig. 7c); hence has implicitly disclosed a form of evaluating which event handler to invoke.

As per claim 19, see claim 8 for corresponding rejection.

As per claim 20, Cahill discloses a method of generating code listing from an object code sequence comprising a section data including a plurality of program instructions, said section associated a relocation section including at least one relocation instruction (e.g. *tag table 218, text relocs 305 , data relocs 306* - Fig. 3; col. 7, lines 26-35) used at link time to modify the object code to generate an executable program (Fig. 1 – Note: linker and object code implicitly discloses modify object code at link time to generate executable), the method comprising:

for each location in the section data determining ... (e.g. *relocation tags* - col. 3, lines 18-25; col. 6, line 56 to col. 7, line 35 – Note: based of derived PIOF from scanning a object module section scanning, locating tags for relocation type is equivalent to determining if relocation instruction is associated with a section data);

deriving relocation additional information ... (e.g. *tag type* - col. 7, lines 25-35; *TYPE, NEXT-INST, TAGS* – Fig. 4); and deriving from the type of relocation instruction additional information (e.g. col. 7, lines 31-35; Fig. 4)

generating the reversed object code listing ...(e.g. Fig. 2, 5, 7A-D), as correspondingly addressed in claim 1/section 4 above.

But Cahill does not explicitly specify reading said relocation instruction identified for each location in the section data. Nor does Cahill specify generating the source code listing in lieu of reversed object code for the location in the section data, such listing being derived from said relocation instruction and additional relocation information. But these limitations have been addressed in claim 1 above.

As per claim 21, Cahill discloses a lister for generating reversed object code listing from an object code sequence comprising program instructions associating a relocation instruction therewith; the lister comprising:

a instruction reader (e.g. col. 6, lines 55-65);

a relocation reader (*relocation tags* - col. 3, lines 18-25; col. 6, line 56 to col. 7, line 35); and determining a type of relocation (*tag type* - col. 7, lines 25-35; *TYPE, NEXT-INST, TAGS* – Fig. 4);

a relocation identifier (*relocation tags* - col. 3, lines 18-25; Fig. 5);

a disassembler for disassembling said program instructions to generate reversed object code and disassembling additional information (e.g. *tags* - col. 7, lines 25-35) from said relocation instruction (Fig. 2, 5; 7A-D).

But Cahill does not specify generating the source code display in lieu of reversed object code from disassembling program instructions, such source code listing being derived from said relocation instruction and additional relocation information and being in human-readable form. But this limitation has been addressed in claim 1 above.

5. Claim 1-4, 6-17, 19 and 20-21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272, in view of Noda et al., JPPN: JP408147155 A (now provided with a translated version - hereinafter Noda).

As per claim 1, Cahill discloses a method of generating source code listing from an object code sequence comprising a section data including a plurality of program instructions, said section associated a relocation section including at least one relocation instruction (e.g. *tag table 218, text relocs 305 , data relocs 306* - Fig. 3) used at link time to modify the object code to generate an executable program (Fig. 1 – Note: linker and object code implicitly discloses modify object code at link time to generate executable), the method comprising:

determining (location has a relocation instruction);

deriving (relocation additional information); and

generating (reversed object code), such limitations having been set forth in claim 1 as set forth in section 4 (using Cahill and Brooks) .

However, Cahill does not explicitly specify reading said relocation instruction identified in the section data. But this limitation has been addressed in claim 1 as set forth in section 4 (using Cahill and Brooks).

Nor does Cahill specify generating the source code listing in lieu of reversed object code for the location in the section data, such listing being derived from said relocation instruction and additional relocation information. Using labels for additional information similar to Cahill's tag table, Noda, in a method to disassemble assembler code, discloses utilizing relocation instruction to inversely assemble source code based on the modification performed when using those labels addresses (Abstract; Constitution). The fact of replacing instruction data represented by tag or

labels with the actual location value by Noda is equivalent in function to the relocation instruction performing offset address replacement as known in the art. Therefore, it would have been obvious for one of ordinary skill in the art at the time the invention was made to implement the disassembling process as taught by Cahill so that from the results provided in the disassembling stage, code can be reversed into source code representation according to the replacement scheme using object code instructions in association of labels and symbol table as taught by Noda. The motivation would have been the same as put forth in claim 1 of section 4 above (using Cahill and Brooks)

As per claim 2, this limitation would have been obvious in view of the rationale used in claim 1 above using Noda's source code teachings.

As per claims 3-4, see rejection in claims 3-4 from USC 103(a) section from above using Cahill and Brooks (Note: Brooks and Noda bear similarity in reconstructing code by replacement of token or labels).

As per claim 6, see claim 6 rejection from USC 103(a) section from above using Cahill and Brooks.

As per claims 7-10, see respective claims 7-10 rejections from USC 103(a) section from above using Cahill and Brooks.

As per claim 11, Cahill discloses a lister for generating reversed object code listing from an object code sequence comprising program instructions associating a relocation instruction therewith; the lister comprising:

a instruction reader (e.g. col. 6, lines 55-65);

a relocation reader (*relocation tags* - col. 3, lines 18-25; col. 6, line 56 to col. 7, line 35);

a relocation identifier (*relocation tags* - col. 3, lines 18-25; Fig. 5);

a disassembler for disassembling said program instructions to generate reversed object code and disassembling additional information (e.g. *tags* - col. 7, lines 25-35) from said relocation instruction (Fig. 2, 5,7A-D).

But Cahill does not specify generating the source code display in lieu of reversed object code listing from disassembling program instructions, such source code listing being derived from said relocation instruction and additional relocation information and being in human-readable form. But this limitation has been addressed in claim 1 from USC 103(a) section 5 from above (using Cahill and Noda).

As per claims 12-17, and 19, see respective claims 12-17, and 19 rejections from USC 103(a) section from above using Cahill and Brooks.

As per claims 20 and 21, these claims disclose the same limitations as addressed in claim 20 and 21 of section 4 above, respectively; and are rejected herein with the corresponding rejections as set forth in claim 20 and 21 section 4, respectively.

For claim 20, Cahill does not explicitly specify reading said relocation instruction identified for each location in the section data. Nor does Cahill specify generating the source code listing in lieu of reversed object code for the location in the section data, such listing being derived from said relocation instruction and additional relocation information. But these limitations have been addressed in claim 20 section 4 above.

For claim 21, Cahill does not specify generating the source code display in lieu of reversed object code from disassembling program instructions, such source code listing being

derived from said relocation instruction and additional relocation information and being in human-readable form. But this limitation has been addressed in claim 21 section 4 above.

6. Claim 5 is rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272 and Brooks et al., USPN: 5,819,097; as applied to claim 1/section 4; and further in view of Brandes, USPN: 5,946,484 (hereinafter Brandes).

As per claim 5, Cahill does not specify a library of object code holding a number of frequently used code sequences together with their associated relocations. But Cahill discloses means for preventing contention in re-modifying already modified libraries (library 170 – Fig. 1; col. 15, lines 12-45) and Brooks discloses an instruction table to expedite reallocation of instructions data at runtime (Fig. 3,5); thus suggesting a means to alleviate extraneous usage of resources by avoiding re-processing of entries or instructions already addressed or scanned during the linking or assembling process, i.e. marking objects, entries/steps so that extraneous re-processing of those objects or entries/steps is a well-known concept in computer processing, when such processing involves iterative or highly repetitive executing steps. Likewise, in a method to regenerate source program from an assembled object code format analogous to the method by Cahill/Brooks, Brandes discloses processing a plurality of object code format instructions and comparing them against pre-recorded patterns (e.g. col. 13, lines 37-64) and elimination of superfluous generation of source file (*NOCMT*- col. 27, lines 23-55). The recording of frequently encountered patterns of object code instructions and avoiding of recreating of superfluous code is suggested. In view of such teachings by Brandes, it would have been obvious for one of ordinary skill in the art at the time the invention was made to modify the object module storage associated with relocation information by Cahill/Brooks so to additionally

store object code in library for reuse using the concept of storing most frequently used patterns of code as suggested by Brandes. One of ordinary skill in the art would be motivated to do so because that way frequently encountered object code format and instruction structures are stored in reusable library form would alleviate extraneous use of resources as observed above.

7. Claim 5 is rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272 and Noda et al., JPPN: JP408147155 A; as applied to claim 1/section 5; and further in view of Brandes, USPN: 5,946,484.

As per claim 5, Noda discloses using tags to expedite processing of object code during disassembling process (Abstract; Constitution) and Cahill discloses alleviating resources contention (col. 15, lines 12-45), both hence suggesting marking objects, entries/steps so that extraneous re-processing of those objects or entries/steps, a well-known concept in computer processing, when such processing involves iterative or highly repetitive executing steps. Further, Brandes discloses processing a plurality of object code format instructions and comparing them against pre-recorded patterns (e.g. col. 13, lines 37-64) and elimination of superfluous generation of source file (*NOCMT*- col. 27, lines 23-55). The limitation to include most frequently used code in library object code is rejected herein using the rationale for combining with Brandes for the same motivation as set forth in claim 5 from USC 103(a) section from above using Cahill and Brooks.

8. Claim 18 is rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272 and Brooks et al., USPN: 5,819,097, as applied to claim 17/section 4; and further in view of Clarke et al., USPN: 5,754,759 (hereinafter Clarke).

As per claim 18, Cahill discloses tag stack and event handler selection as addressed in section 4 claims 6 and 17; but does not explicitly discloses event stack. Clarke, in a method to monitor program debug execution using disassembly module (col. 6, line 12-25) similar to Cahill's error-check of object modules at run-time, discloses a event stack (col. 9, lines 49-67). In view of the teachings by Cahill from above, it would have been obvious for one of ordinary skill in the art at the time the invention was made to modify the use of stack and signal handler by Cahill so to implement an event stack as taught by Clarke because use of stack as taught by Clarke for storage and immediate resolution of instantaneous occurring of signal handler requests as suggested by Cahill would enable fulfilling of a larger amount of distress calls incurred in a system where error-checking involving memory address readjustment is as crucial as suggested in the relocation execution by Cahill.

9. Claim 18 is rejected under 35 U.S.C. 103(a) as being unpatentable over Cahill et al., USPN: 6,021,272 and Noda et al., JPPN: JP408147155 A, as applied to claim 17/section 5; and further in view of Clarke et al., USPN: 5,754,759.

As per claim 18, see rationale of claim 18 above from 35 U.S.C. 103(a) with Cahill et al. and Brooks et al., USPN: 5,819,097, in view of Clarke from above.

Response to Arguments

10. Applicant's arguments filed 3/29/2004 have been fully considered but they are not persuasive. Following are Examiner's observations therefor.

(A) Applicant submitted that Cahill 'is not concerned with a lister ... no disclosure ... for reading relocation instructions ... PIAI module ... assembly code is not source code ... tags are provided from ... PIOF module ... discloses a disassembling process to recompose the original

object code ... new object module 150' (Appl. Rmrks, pg. 9, bottom 2 para; pg. 10, top 2 para). In response, it is pointed out in the rejection how each limitation has a corresponding portion in Cahill for matching the feature such as section data having relocation instruction, reading such instruction, deriving additional information; and generating disassembled object code. A instruction being encoded in forms a tag number is interpreted as a instruction and the fact that it corresponds with relocating of data and instructions in relation to data associated with the object module and its corresponding symbol table makes it an instruction for relocation found in processing the object module sections. The claims do not provide specific description of what a relocation instruction consists of; nor do the claims specify what exactly is consisting the so-called additional information derived from those relocation instructions. The rejection has pointed out that by differentiating the relocation-related tag (or instruction), additional information such as meaning or contents of pointer structures correlating the type of tags can be determined (Fig. 4). As for the object code disassembling instead of listing of a source code, the rejection has pointed out how by using the same process of storing elements from a object code into tables or labels, the analogous approach by Brooks can be joined with the findings by Cahill for using the same stored pointers and data structures in order to go from disassembling back into a source code. Whether to convert a disassembled code to another newer object code or to a source code, either form corresponds to the data being persisted via different means such as pointer, linked lists, table or tree of symbols as taught by Brooks and Cahill; and it would be obvious mostly because the manner in which those object code elements are stored is similar to both references. Besides, the disassembling of an object code to make new re-assembled code is not very distant from disassembling an object code back into its source code form because

disassembling is one crucial of decompiling, and the results from such disassembling can serve as resources to attain either what Cahill intended or what Brooks intended. As long as data are structured and stored for the purpose to reconstruct code which has been disassembled, whether this code reconstructing yields object code or source code is not an effort that would require undue experimentation. The effort here being the techniques that have been provided during the course of disassembling, and reconstructing a object code or a source code is barely an intended use from the resources made available upon the above techniques. In regard to which, Both Cahill (Fig. 4,5, 7A) and Brooks (Fig. 5) use tables, linked objects or symbol referencing and replacing as well as tokens or tags to represent temporary data prior to linking or relinking time usage. In contrast, the invention as claimed does not specify the steps that enable the use of additional information in conjunction with the 'relocation instructions' to effect the source code listing; and this is open to the broadest interpretation. The 103(a) rejection has laid out how it would have been obvious to go from Cahill to Brooks because of the similarity in both references to combine data stored prior to decompilation or reassembling, with the motivation as to how having a source code would make it more flexible to reuse a reverse-engineered code in other applications.

(B) Applicant has submitted that Brooks does not mention about 'memory address reallocated values' but merely teaches that the '... that is the tags and symbol table ... are merely used for establishing ... operand and associated variable name ... an operand variable name and location is ... not the same as relocation instructions' (Appl. Rmrks, pg. 11, 2nd para, pg. 12, top para). As addressed in section A above, Brooks is used because provide a tag or label replacing of variable or operands found in the linking process; and such replacement at linking time is

reminiscent of a generic relocation operation as well as analogous to the tagging of data in association with relocation and symbol table during the processing of a object code section by Cahill. These techniques of keeping track of how data are persisted and thereby can be recovered at reassembling time are similar and the motivation as to use Cahill's techniques combined with the approach to decompile into human-readable source code has been addressed in the rejection. Again, it is noted that the claim does not specify what a relocation instruction is all about and what underlies the mechanism as to make use of derived additional information from the section data and those very instructions in order to arrive at the listing of source code as purported by the invention. The rejection has shown how much Cahill has met the features required in the claim, and how the approach by Brooks can be joined with Cahill in order to make the extra step as to deliver a listing of source code from disassembled code elements because Cahill and Brooks do share one thing in common: how to decompose an object code using a symbol table and instruction table or relocation/address referencing information in order to reconstruct them back using the replacement techniques upon temporary variable holders such as tag table or tokens.

(C) As per claims 5 and 18, refer to sections A and B above.

(D) Applicant has submitted that Examiner has used Cahill as opposed to Brandes in the mis-stated office action header relating the USC 35 103(a) combining Cahill with Noda (pg. 13, 1st and 2nd para). The current rejection has now been readjusted to point how Noda's teaching is similar to Cahill's so to be combined thereto in order to provide source code restoration. In response to a point bought forth by Applicant (pg. 13, last para), the point that Noda appears to be similar to Brooks only emphasizes how Noda can be combined with Cahill for the same

rationale as set forth in the combination Cahill/Brooks in the upper part of the rejection. The claim does not specify how the relocation instructions identified from the object code are constructed so to convey in what specific way do they distinguish from operations disclosed by Noda as to replace a referred to variable location with the correct dynamic value thereof at linking time, i.e. relocation operation (see Noda's Background discussion). Applicant's assertion on the use of symbol table is not persuasive so as to establish that what is claimed by the invention differs from Noda or Brooks' technique of replacing earlier tags, tokens or labels with the actual value at link/relink time; because the claim as recited does not enforce that the use of relocation instructions in conjunction with the section data has to be done after the symbol table has been put to rest. Thus, Applicant has failed to point out how the combination of Cahill and Noda results in an inappropriate amalgaming, or teaches away from their respective intents. And as long as the use of symbol table along with other means of storage enable relocation operations at re-assembling time, the very end results deriving from such forms of relocation operations are reading on the claims and will be used. As claimed, the invention omits essentials steps as to enable the features recited as 'relocation instructions' in conjunction with some information derived from reading a section data to specifically reconstruct the original source code whose information have been lost during the process of assembling into the very object code. And as such, it is likely that the invention is being characterized as non-enabling for one skill in the art; and the non-enabling status of this sort has its merit potentially leading to specific type of rejections practiced in the Office.

(E) Applicant has submitted that a relocation instruction is not same as a symbol table (Appl. Rmrks, pg. 14, middle). Applicant is referred back to section D above.

Thus, the rejection will stand as is.

Conclusion

11. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (703)305-7207. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (703)305-9662.

Any response to this action should be mailed to:

Commissioner of Patents and Trademarks

Washington, D.C. 20231

or faxed to:

(703) 872-9306 (for formal communications intended for entry)

or: (703) 746-8734 (for informal or draft communications, please consult Examiner before using this number)

Hand-delivered responses should be brought to Crystal Park II, 2121 Crystal Drive, Arlington. VA. , 22202. 4th Floor(Receptionist).

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

VAT
May 24, 2004

Kakali Chaki

KAKALI CHAKI
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100